Greedy Partial Knapsack, Greedy Huffman Coding

Week-05, Lecture-02

Course Code: CSE221

Course Title: Algorithms

Program: B.Sc. in CSE

Course Teacher: Tanzina Afroz Rimi

Designation: Lecturer

Email: tanzinaafroz.cse@diu.edu.bd

Knapsack Problem

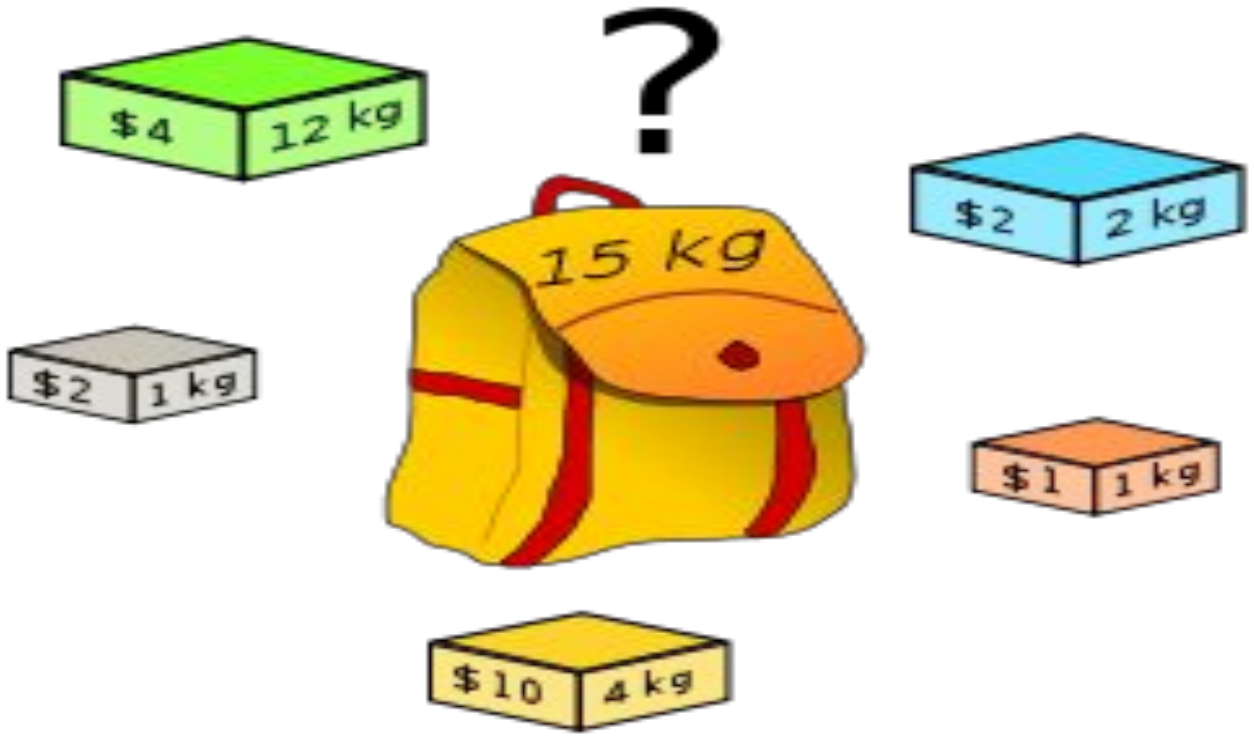
Knapsack problem

A 1998 study of the Stony Brook University Algorithm Repository showed that, out of 75 algorithmic problems, the knapsack problem was the 18th most popular and the 4th most needed after kd-trees, suffix trees, and the bin packing problem

Knapsack Problem

- Given a set of items, each with a mass and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
- It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

Knapsack Problem



which boxes should be chosen to maximize the amount of money while still keeping the overall weight under or equal to 15 kg?

Answer: **3 yellow boxes** and **3 grey boxes**

Knapsack Problem

- In a ***knapsack problem*** or ***rucksack problem***, we are given a set of n items, where each item i is specified by a size s_i and a value v_i. We are also given a size bound S, the size of our knapsack.

Item #	Size	Value
1	1	8
2	3	6
3	5	5

Knapsack Problem

There are two versions of the problem:

1. 0-1 Knapsack Problem
2. Fractional Knapsack Problem
   i. Bounded Knapsack Problem
   ii. Unbounded Knapsack Problem

Solutions to Knapsack Problems

- Greedy Algorithm – keep taking most valuable items until maximum weight is reached or taking the largest value of each item by calculating $V_i = \frac{value_i}{size_i}$
- Dynamic Programming – solve each sub problem once and store their solutions in an array

Points to Remember

- In this problem we have a knapsack that has a weight limit W.
- There are items $i_{1,}$ $i_{2,}$ ……i_n each having weight w_1,w_2…..w_n and some benefit(value or profit) associated with it $v_{1,}$ v_2 ,…….$v_{n.}$
- Our objective is to maximize the benefit such that the total weight inside the knapsack is at most W.
- And we are also allowed to take an item in fractional part.

Assume that we have a knapsack with max weight capacity

$$W=16$$

Our objective is to fill the knapsack with items such that the(value or profit) is maximum.

Consider the following items and their associated weight and value. W=16

ITEM	WEIGHT	VALUE
i_1	6	6
i_2	10	2
i_3	3	1
i_4	5	8
i_5	1	3
i_6	3	5

Steps

Calculate value per weight for each item(we can call this value density).

Sort the items as per the value density in descending order.

Take as much as possible not already taken in the knapsack.

Calculate density
W=16

ITEM	WEIGHT	VALUE	DENSITY
i_1	6	6	1.00
i_2	10	2	0.200
i_3	3	1	0.333
i_4	5	8	1.600
i_5	1	3	3.000
i_6	3	5	1.667

Now we will pick items such that our benefit is maximum and total weight of the selected items is at most W.
W=16

ITEN	WEIGHT	VALUE	DENSITY
i_5	1	3	3.000
i_6	3	5	1.667
i_4	5	8	1.600
i_1	6	6	1.000
i_3	3	1	0.333
i_2	10	2	0.200

Now we draw another table to represent knapsack. Our objective is to fill the knapsack with items to get maximum benefit without crossing weight limit W=16.

ITEN	WEIGHT	VALUE	DENSITY
i_5	1	3	3.000
i_6	3	5	1.667
i_4	5	8	1.600
i_1	6	6	1.000
i_3	3	1	0.333
i_2	10	2	0.200

Knapsack

ITEM	WEIGHT	VALUE	TOTAL WEIGHT	BENIFIT

W=16 check i_5 is WEIGHT(i_5)+TOTAL WEIGHT<=W

1+0<=16

1<=16

YES

So we take the whole item.

Total weight inside knapsack is 0

Knapsack

ITEN	WEIGHT	VALUE	DENSITY
i_5	1	3	3.000
i_6	3	5	1.667
i_4	5	8	1.600
i_1	6	6	1.000
i_3	3	1	0.333
i_2	10	2	0.200

ITEM	WEIGHT	VALUE	TOTAL WEIGH	BENIFIT
i_5	1	3	1.000	3.000

W=16 check i_6 is WEIGHT(i_6)+TOTAL WEIGHT<=W

3+1<=16

4<=16

YES

So we take the whole item.

Total weight inside knapsack is 1.000

Knapsack

ITEN	WEIGHT	VALUE	DENSITY
i_5	1	3	3.000
i_6	3	5	1.667
i_4	5	8	1.600
i_1	6	6	1.000
i_3	3	1	0.333
i_2	10	2	0.200

ITEM	WEIGHT	VALUE	TOTAL WEIGH	BENIFIT
i_5	1	3	1.000	3.000
i_6	3	5	4.000	8.000

W=16 check i_4 is WEIGHT(i_4)+TOTAL WEIGHT<=W

5+4<=16

9<=16

YES

So we take the whole item.

Total weight inside knapsack is 4.000

Knapsack

ITEN	WEIGHT	VALUE	DENSITY
i_5	1	3	3.000
i_6	3	5	1.667
i_4	5	8	1.600
i_1	6	6	1.000
i_3	3	1	0.333
i_2	10	2	0.200

ITEM	WEIGHT	VALUE	TOTAL WEIGH	BENIFIT
i_5	1	3	1.000	3.000
i_6	3	5	4.000	8.000
i_4	5	8	9.000	16.000

W=16 check i_1 is WEIGHT(i_1)+TOTAL WEIGHT<=W

6+9<=16

15<=16

YES

So we take the whole item.

Total weight inside knapsack is 9.000

Knapsack

ITEN	WEIGHT	VALUE	DENSITY
i_5	1	3	3.000
i_6	3	5	1.667
i_4	5	8	1.600
i_1	6	6	1.000
i_3	3	1	0.333
i_2	10	2	0.200

ITEM	WEIGHT	VALUE	TOTAL WEIGH	BENIFIT
i_5	1	3	1.000	3.000
i_6	3	5	4.000	8.000
i_4	5	8	9.000	16.000
i_1	6	6	15.000	22.000

W=16 check i_3 is WEIGHT(i_3)+TOTAL WEIGHT<=W

3+15<=16

18<=16

NO

So we take the fractional part.

Total weight inside knapsack is

15.000

Knapsack

ITEN	WEIGHT	VALUE	DENSITY
i_5	1	3	3.000
i_6	3	5	1.667
i_4	5	8	1.600
i_1	6	6	1.000
i_3	3	1	0.333
i_2	10	2	0.200

ITEM	WEIGHT	VALUE	TOTAL WEIGH	BENIFIT
i_5	1	3	1.000	3.000
i_6	3	5	4.000	8.000
i_4	5	8	9.000	16.000
i_1	6	6	15.000	22.000

We will fill the knapsack with 1 weight of i_3 item having value(1/3=0.333)

ITEN	WEIGHT	VALUE	DENSITY
i_5	1	3	3.000
i_6	3	5	1.667
i_4	5	8	1.600
i_1	6	6	1.000
i_3	3	1	0.333
i_2	10	2	0.200

Knapsack

ITEM	WEIGHT	VALUE	TOTAL WEIGH	BENIFIT
i_5	1	3	1.000	3.000
i_6	3	5	4.000	8.000
i_4	5	8	9.000	16.000
i_1	6	6	15.000	22.000
i_3	1	0.333	16.000	22.333

W=16
Total weight inside the knapsack is 16
so we stop here
TOTAL WEIGHT inside the knapsack is 16.000

ITEN	WEIGHT	VALUE	DENSITY
i_5	1	3	3.000
i_6	3	5	1.667
i_4	5	8	1.600
i_1	6	6	1.000
i_3	3	1	0.333
i_2	10	2	0.200

Knapsack

ITEM	WEIGHT	VALUE	TOTAL WEIGH	BENIFIT
i_5	1	3	1.000	3.000
i_6	3	5	4.000	8.000
i_4	5	8	9.000	16.000
i_1	6	6	15.000	22.000
i_3	1	0.333	16.000	22.333

So the maximum benefit is 22.333

Knapsack

ITEN	WEIGHT	VALUE	DENSITY
i_5	1	3	3.000
i_6	3	5	1.667
i_4	5	8	1.600
i_1	6	6	1.000
i_3	3	1	0.333
i_2	10	2	0.200

ITEM	WEIGHT	VALUE	TOTAL WEIGH	BENIFIT
i_5	1	3	1.000	3.000
i_6	3	5	4.000	8.000
i_4	5	8	9.000	16.000
i_1	6	6	15.000	22.000
i_3	1	0.333	16.000	22.333

Practice problem

Items:

- Item 1: Weight = 3, Value = 24
- Item 2: Weight = 6, Value = 30
- Item 3: Weight = 2, Value = 14
- Item 4: Weight = 5, Value = 20
- Item 5: Weight = 4, Value = 18

Knapsack Capacity: 10

Greedy Algorithm

- The optimal solution to the fractional knapsack
- Not an optimal solution to the 0-1 knapsack

Huffman Codes

Huffman Codes

- Widely used technique for data compression
- Assume the data to be a sequence of characters
- Looking for an effective way of storing the data
- ***Binary character code***
  - Uniquely represents a character by a binary string

Fixed-Length Codes

E.g.: Data file containing 100,000 characters

	a	b	c	d	e	f
Frequency (thousands)	45	13	12	16	9	5

- 3 bits needed
- a = 000, b = 001, c = 010, d = 011, e = 100, f = 101
- Requires: 100,000 $\cdot$ 3 = 300,000 bits

Huffman Codes

- Idea:
  - Use the frequencies of occurrence of characters to build a optimal way of representing each character

	a	b	c	d	e	f
Frequency (thousands)	45	13	12	16	9	5

Variable-Length Codes

E.g.: Data file containing 100,000 characters

	a	b	c	d	e	f
Frequency (thousands)	45	13	12	16	9	5

- Assign short codewords to frequent characters and long codewords to infrequent characters
- a = 0, b = 101, c = 100, d = 111, e = 1101, f = 1100
- $(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1{,}000$

  $= 224{,}000$ bits

Prefix Codes

- Prefix codes:
  - Codes for which no codeword is also a prefix of some other codeword
  - Better name would be “prefix-free codes”
- We can achieve optimal data compression using prefix codes
  - We will restrict our attention to prefix codes

Encoding with Binary Character Codes

- Encoding
  - Concatenate the codewords representing each character in the file
- *E.g.*:
  - a = 0, b = 101, c = 100, d = 111, e = 1101, f = 1100
  - abc = 0 · 101 · 100 = 0101100

Decoding with Binary Character Codes

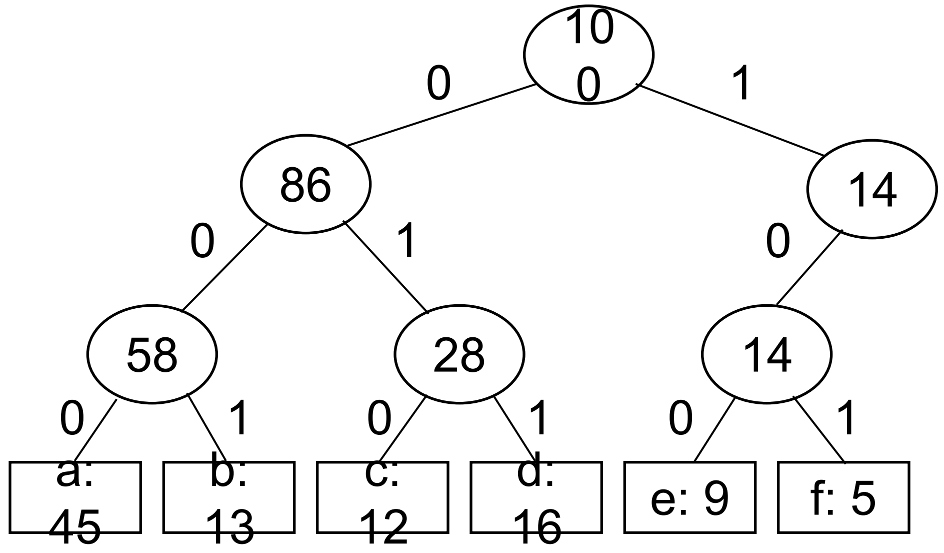
- Prefix codes simplify decoding
  - No codeword is a prefix of another ⇒ the codeword that begins an encoded file is unambiguous
- Approach
  - Identify the initial codeword
  - Translate it back to the original character
  - Repeat the process on the remainder of the file
- *E.g.*:
  - a = 0, b = 101, c = 100, d = 111, e = 1101, f = 1100
  - 001011101 =

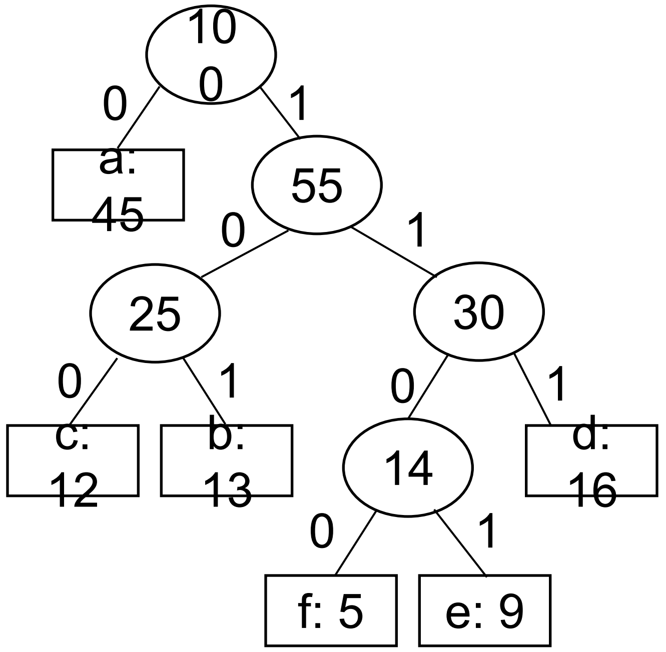
0 · · · = aabe

0 101 1101

Prefix Code Representation

- Binary tree whose leaves are the given characters
- Binary codeword
  - the path from the root to the character, where 0 means “go to the left child” and 1 means “go to the right child”
- Length of the codeword
  - Length of the path from root to the character leaf (depth of node)





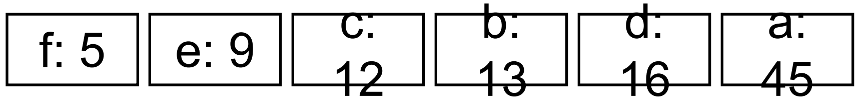
Optimal Codes

- An optimal code is always represented by a **full binary tree**
  - Every non-leaf has two children
  - Fixed-length code is not optimal, variable-length is
- How many bits are required to encode a file?
  - Let C be the alphabet of characters
  - Let $f(c)$ be the frequency of character c
  - Let $d_T(c)$ be the depth of c's leaf in the tree T corresponding to a prefix code

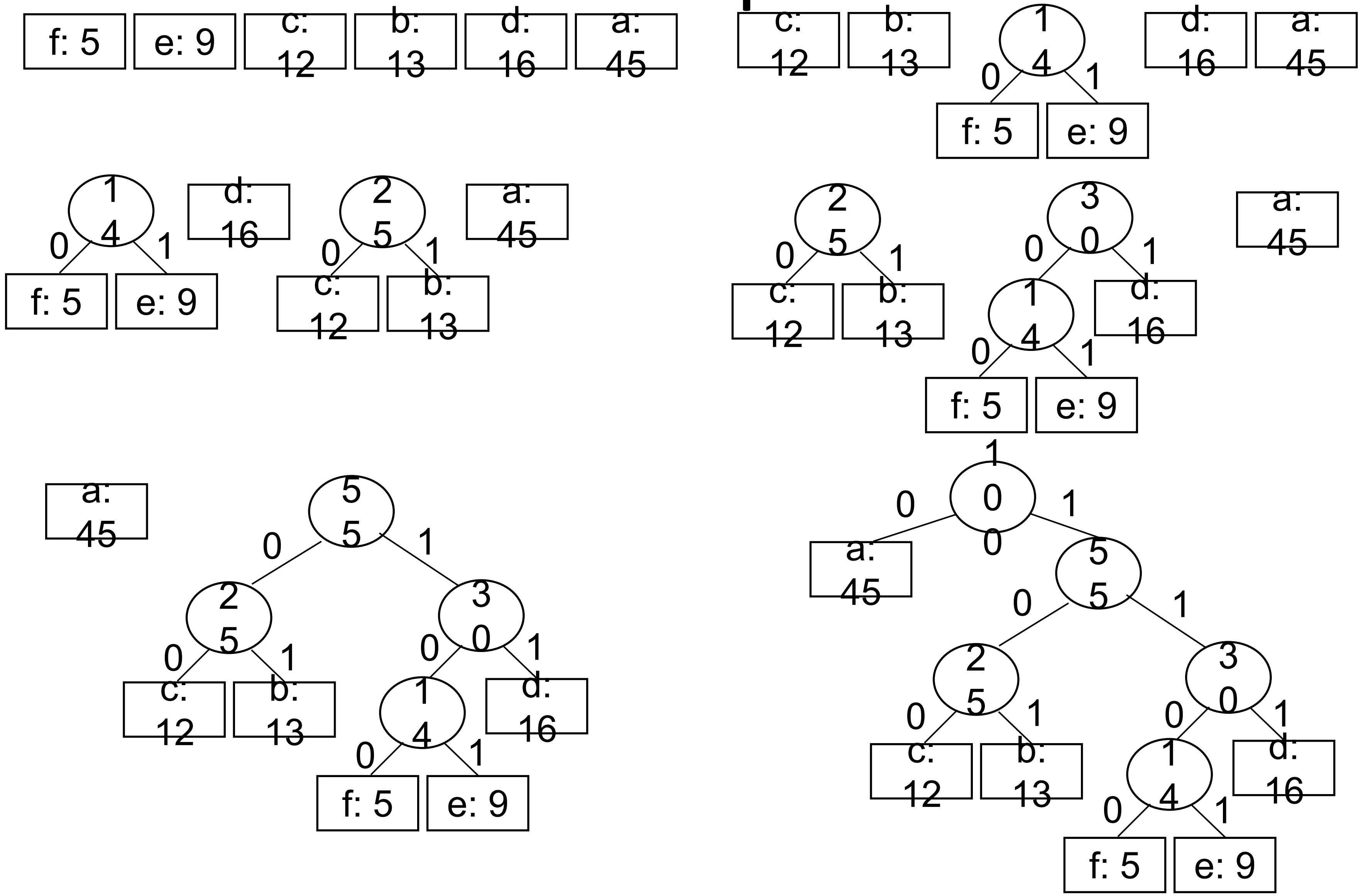
$$B(T) = \sum_{c \in C} f(c) d_T(c)$$ the cost of tree T

Constructing a Huffman Code

- A greedy algorithm that constructs an optimal prefix code called a **Huffman code**
- Assume that:
  - C is a set of n characters
  - Each character has a frequency $f(c)$
  - The tree T is built in a bottom up manner
- Idea:
  - Start with a set of $|C|$ leaves
  - At each step, merge the two least frequent objects: the frequency of the new node = sum of two frequencies
  - Use a min-priority queue Q, keyed on f to identify the two least frequent objects



Example



Textbooks & Web References

- Text Book (Chapter 16)
- Reference book iii (Chapter 17)
- www.geeksforgeeks.org
- www.codeforces.com

Thank you

&

Any question?